Final Project Report | OOAD
Justin Astalos and Benjamin Kohav

LINK TO DEMO: FinalProjectDemo.mp4

1. **Name of Project and Team Members**

Fight Boat (Battleship) – by Benjamin Kohav and Justin Astalos

2. **Final State of System Statement**

A paragraph on the final state of your system: what features were implemented, what features were not and why, what changed from Project 5 and 6

The final state of our system contains the following features: A Battleship digital simulation with an interactive User Interface plus some additional features to make the game more interesting: different game boards for users to play with (e.g. Diamond Board, Smiley Board), different boat fleets for users to select from (e.g. Attack Fleet, Economic Fleet), and finally a monetary aspect of the game in which some of your boats could make money for you and you can use that money to upgrade some of your ships for better/more lethal attacks on your opponent's board. The features that were not implemented were making the boats vertical (not just horizontal as is currently implemented, using an AI to act as the second player in the game, and having a better transition in between players so that the screen change isn't as jarring. These features were not implemented because of a lack of time, after 100+ combined hours from our team, this is what we were able to produce.

3. **Final Class Diagram and Comparison Statement**

A thorough UML class diagram representing your final set of classes and key relationships of the system. Highlight and document in that diagram any patterns that were included (in whole or part) in your design. Include the class diagram submitted in Project 5, and use it to show what changed in your system from that point into the final submission

Support the diagrams with a written paragraph identifying key changes in your system since your design/work was submitted in Projects 5 and 6

Note: Project UML Diagrams for each phase are added to the final github repo and clearly titled. They will also be added to the end of this project report.

While the overall vision and scope of our project did not change by much, the UML class diagram for the final project compared to the original draft from Project 5 reflects some changes in our approach to the user interface in particular; this is mostly because we did not anticipate some of the shortcomings of using the JFrame Library until later, when we better understood what it was capable of doing and what it was not capable of doing. This included creation of different classes that made special JPanels for the different pages we had, as well as keeping track of the game boards that we are playing with using a class called Square, which stores information about one specific board square from the entire board. This was created in lieu of issues with making changes to the board storage on the actual JComponent Board that we originally envisioned. While most of our patterns that we originally planned to use were incorporated into the final project (Observer, Singleton, and Strategy), we ended up changing our last pattern to Command following feedback from our demo in Project 6. The Command pattern was better suited towards our project because we had elements when actions were enabled the same way on our user interface but the responses were dependent on circumstances, which the command pattern handles perfectly.

4. **Third-Party code vs. Original code Statement**

A clear statement of what code in the project is original vs. what code you used from other sources– whether tools, frameworks, tutorials, or examples – this section must be present

even if you used NO third-party code - include the sources (URLs) for your third-party elements.

Statement: We did not have any code that was directly related to how we put together the final product for our project, rather, there were bits and pieces of the fundamental functionality of our code (particularly stemming from creation of JFrames for each page of our user interface) where we had to utilize a base code set up for each JFrame and the panels that make up each JFrame display to the game players. These code guidelines came directly from the library descriptions of JFrame/Swing and some individual internet posts that clarify how to use parts of the library to create a certain "look" or aesthetic for your JFrames. See the links below for what was referenced in the creation of this project.

Sources for third party elements:

https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html

https://stackoverflow.com/questions/7696347/to-break-a-message-in-two-or-more-lines-in-joptionpane/7707397#7707397

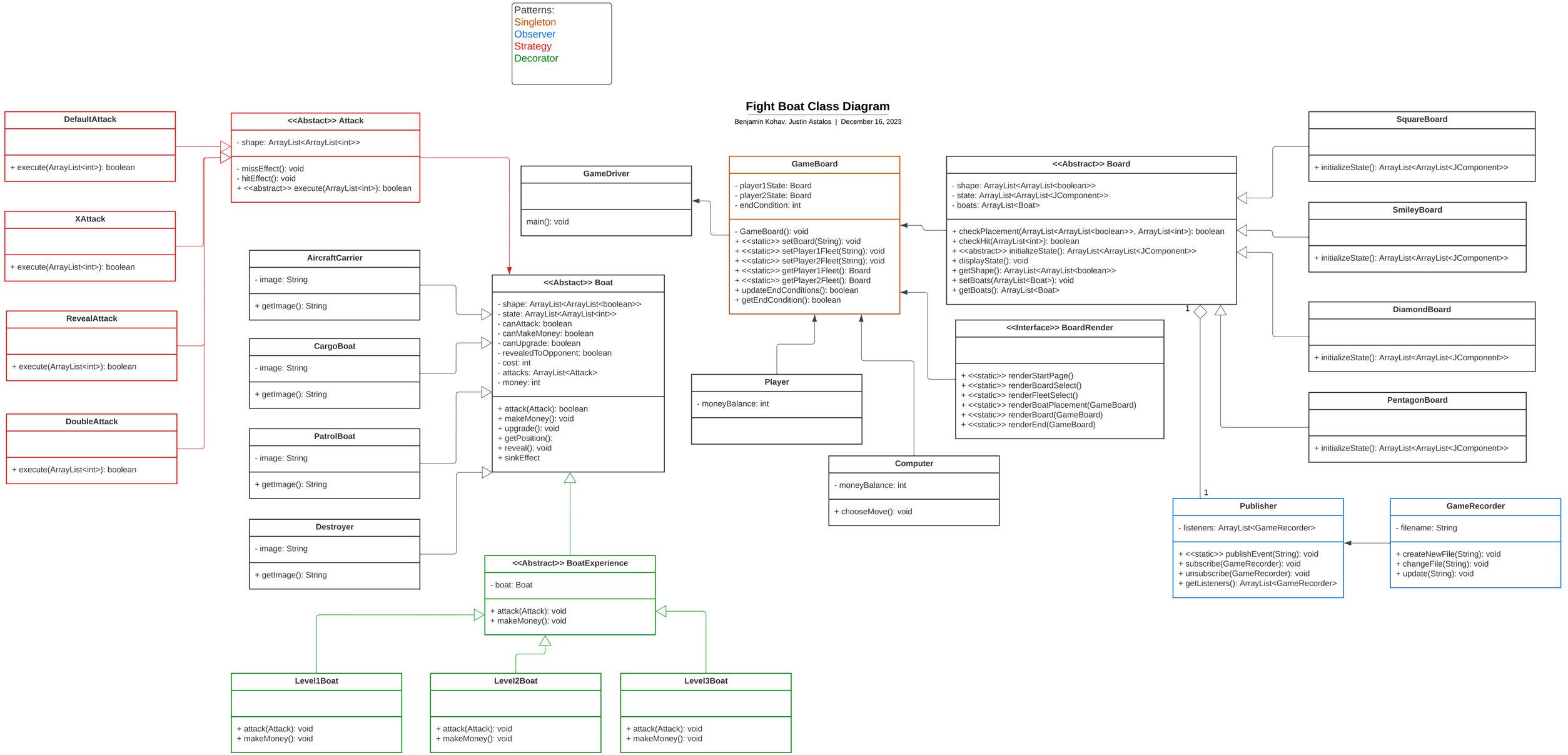
https://java-demos.blogspot.com/2012/09/setting-background-image-in-jframe.html

https://stackoverflow.com/questions/1064977/setting-background-images-in-jframe

5. **Statement on the OOAD process for your overall Semester Project**

List three key design process elements or issues (positive or negative) that your team experienced in your analysis and design of the OO semester project

1. Issue: Understanding the set up and framework uses for our User Interface Library, JFrame.
2. Element: Utilization of the patterns as early as possible made development down the line substantially easier
3. Element: Continuous communication and frequent meetings helped us reach nearly all of our design goals.


Patterns:
Singleton
Observer
Strategy
Decorator
Fight Boat Class Diagram
Benjamin Kohav, Justin Astalos | December 16, 2023
DefaultAttack
+ execute(ArrayList<int>): boolean
XAttack
+ execute(ArrayList<int>): boolean
RevealAttack
+ execute(ArrayList<int>): boolean
DoubleAttack
+ execute(ArrayList<int>): boolean
<<Abstact>> Attack
- shape: ArrayList<ArrayList<int>>
- missEffect(): void
- hitEffect(): void
+ <<abstract>> execute(ArrayList<int>): boolean
GameDriver
main(): void
GameBoard
- player1State: Board
- player2State: Board
- endCondition: int
- GameBoard(): void
+ <<static>> setBoard(String): void
+ <<static>> setPlayer1Fleet(String): void
+ <<static>> setPlayer2Fleet(String): void
+ <<static>> getPlayer1Fleet(): Board
+ <<static>> getPlayer2Fleet(): Board
+ updateEndConditions(): boolean
+ getEndCondition(): boolean
<<Abstract>> Board
- shape: ArrayList<ArrayList<boolean>>
- state: ArrayList<ArrayList<JComponent>>
- boats: ArrayList<Boat>
+ checkPlacement(ArrayList<ArrayList<boolean>>, ArrayList<int>): boolean
+ checkHit(ArrayList<int>): boolean
+ <<abstract>> initializeState(): ArrayList<ArrayList<JComponent>>
+ displayState(): void
+ getShape(): ArrayList<ArrayList<boolean>>
+ setBoats(ArrayList<Boat>): void
+ getBoats(): ArrayList<Boat>
SquareBoard
+ initializeState(): ArrayList<ArrayList<JComponent>>
SmileyBoard
+ initializeState(): ArrayList<ArrayList<JComponent>>
DiamondBoard
+ initializeState(): ArrayList<ArrayList<JComponent>>
PentagonBoard
+ initializeState(): ArrayList<ArrayList<JComponent>>
AircraftCarrier
- image: String
+ getImage(): String
CargoBoat
- image: String
+ getImage(): String
PatrolBoat
- image: String
+ getImage(): String
Destroyer
- image: String
+ getImage(): String
<<Abstact>> Boat
- shape: ArrayList<ArrayList<boolean>>
- state: ArrayList<ArrayList<int>>
- canAttack: boolean
- canMakeMoney: boolean
- canUpgrade: boolean
- revealedToOpponent: boolean
- cost: int
- attacks: ArrayList<Attack>
- money: int
+ attack(Attack): boolean
+ makeMoney(): void
+ upgrade(): void
+ getPosition():
+ reveal(): void
+ sinkEffect
<<Interface>> BoardRender
+ <<static>> renderStartPage()
+ <<static>> renderBoardSelect()
+ <<static>> renderFleetSelect()
+ <<static>> renderBoatPlacement(GameBoard)
+ <<static>> renderBoard(GameBoard)
+ <<static>> renderEnd(GameBoard)
Player
- moneyBalance: int
Computer
- moneyBalance: int
+ chooseMove(): void
1
1
Publisher
- listeners: ArrayList<GameRecorder>
+ <<static>> publishEvent(String): void
+ subscribe(GameRecorder): void
+ unsubscribe(GameRecorder): void
+ getListeners(): ArrayList<GameRecorder>
GameRecorder
- filename: String
+ createNewFile(String): void
+ changeFile(String): void
+ update(String): void
<<Abstract>> BoatExperience
- boat: Boat
+ attack(Attack): void
+ makeMoney(): void
Level1Boat
+ attack(Attack): void
+ makeMoney(): void
Level2Boat
+ attack(Attack): void
+ makeMoney(): void
Level3Boat
+ attack(Attack): void
+ makeMoney(): void

Fight Boat Class Diagram

Justin Astalos, Ben Kohav | December 16, 2023

Patterns:
Singleton
Observer
Strategy
Command

Fleet Description
- boats: ArrayList<boats>
addBoat(): void

Fleet Display
- boats: ArrayList<boats>
addBoat(): void

BackgroundPanel
- image(): Image
paintComponent(): void

PlaceFkeetMenu
+ getPlaceFleetMenu(boolean): JPanel

MiddleMenu
+ MiddleMenu()

MainMenu
+ MainMenu

BoardSelect
+ BoardSelect()

FleetSelect
+ FleetSelect()

PlaceFleet
+ PlaceFleet()

PlayGame
+ PlayGame()

<<Abstract>> Board
- boardName: String
- boardDescription: String
- boats: ArrayList<Boat>
- state: ArrayList<ArrayList<Square>>
- currentAttackingBoat: int
- fleetType: boolean
- boatNotSet(): int
+ <<abstract>> initializeState(): ArrayList<ArrayList<JComponent>>
+ displayState2(boolean currPlayer, bool player1): JLayeredPane
+ setBoats(ArrayList<Boat>): void
+ getBoats(): ArrayList<Boat>
+ setBoatPosition(int i, ArrayList<Integer> position): void
+ attackSpace(ArrayList<Integer> coordinates): boolean
+ setSquare(boolean miss, boolean hit, ArrayList<Integer> position): void
+ checkSunkBoats(): void
+ checkEndState(): boolean
+ getState(): ArrayList<ArrayList<Square>>
+ checkValidPlacement(int boatNum, ArrayList<Int> position): boolean
+ getCurrentAttackingBoat(): int
+ iterateAttackingBoat(): void
+ resetAttackingBoat(): void
+ getBoardName(): String
+ getBoardDescription(): String

SquareBoard
+ initializeState(): ArrayList<ArrayList<JComponent>>

SmileyBoard
+ initializeState(): ArrayList<ArrayList<JComponent>>

DiamondBoard
+ initializeState(): ArrayList<ArrayList<JComponent>>

OAttack
+ <<abstract>> execute(bool player1, ArrayList<Int> position): void

XAttack
+ <<abstract>> execute(bool player1, ArrayList<Int> position): void

DefaultAttack
+ <<abstract>> execute(bool player1, ArrayList<Int> position): void

AdjacentAttack
+ <<abstract>> execute(bool player1, ArrayList<Int> position): void

<<Abstact>> GameAction
- name: String
- imagePath: String
+ <<abstract>> execute(bool player1, ArrayList<Int> position): void
+ getImagePath(): String

Make50
- moneyPerRound: int
+ <<abstract>> execute(bool player1, ArrayList<Int> position): void

Make100
- moneyPerRound: int
+ <<abstract>> execute(bool player1, ArrayList<Int> position): void

GameDriver
main(): void

GameBoard
- <<static>> gameBoard: GameBoard
- player1Board: Board
- player2Board: Board
- gameState: int
- <<static>> numTurn: int
- <<static>> currentFrame: JFrame
- GameBoard(): void
+ <<static>> setBoard(String): void
+ <<static>> setPlayer1Fleet(String): void
+ <<static>> setPlayer2Fleet(String): void
+ <<static>> getPlayer1Board(): Board
+ <<static>> getPlayer2Boar(): Board
+ <<static>> setPlayer1BoatPosition(int i, ArrayList<Integer> position): void
+ <<static>> setPlayer2BoatPosition(int i, ArrayList<Integer> position): void
+ getGameState(): int
+ setGameState(int state): void
+ <<static>> setBoardSquare(bool player1, ArrayLIst<Int> position)>: void
+ <<static>> checkValidPlacement(bool player1, int boatNum, ArrayList<Int> position): boolean
+ <<static>> iterateAttackingBoat(bool player1): void
+ <<static>> resetAttackingBoat(bool player1): void
+ <<static>> addTurn(): void
+ <<static>> getTurn(): int
+ <<static>> setFrame(JFrame currFrame): void
+ <<static>> getFrame(): JFrame
+ <<static>> resetFrame(): void

EndGame
+ EndGame()

Square
- isHit: boolean;
- isMiss: boolean;
- sButton: boolean;
- position: ArrayList<Integer>;
+ generatateButton(Boolean currPlayer, ArrayList<Boat> boats): JButton
+ getColor(boolean currPlayer, ArrayList<Boat> boats): Color
+ checkForBoat(ArrayList<Boat> boats): boolean
+ getHit(): boolean
+ getMiss: boolean
+ returnButton: boolean
+ setHit(boolean isHit): void
+ setMiss(boolean isMiss): void

<<Abstact>> Boat
- boatName: String
- boatDescription: String
- boatUpgradeDescribe: String
- horImagePath: String
- vertImagePath: String
- action: GameAction
- rootPosition: ArrayList<Integer>
- allPositions: ArrayList<ArrayList<Integer>>
- shape: ArrayList<ArrayList<String>>
- isSunk: boolean
+ <<abstract>> getShape(): ArrayList<ArrayList<String>>
+ getHorImagePath(): String
+ getVertImagePath(): String
+ checkHit(ArrayList<Int> coordinates): boolean
+ setPosition(ArrayList<Int> position): void
+ generateAllPositions(ArrayList<Int> position): ArrayList<ArrayList<Int>>
+ updateSunk(ArrayList< ArrayList<Square> > state): void
+ getSpaceImagePath(ArrayList<Integer> Position): String
+ getSunk(): boolean
+ getAction(): GameAction
+ getPosition(): ArrayList<Integer>
+ getBoatName(): String
+ getBoatDescription(): String
+ setBoatUpgradeDescribe(String update): void
+ getBoatUpgradeDescribe(): String

AircraftCarrier
+ getShape(): ArrayList<ArrayList<String>>

CargoBoat
+ getShape(): ArrayList<ArrayList<String>>

PatrolBoat
+ getShape(): ArrayList<ArrayList<String>>

Destroyer
+ getShape(): ArrayList<ArrayList<String>>

Battleship
+ getShape(): ArrayList<ArrayList<String>>

Invoker
- invoker: Invoker
+ <<static>> execute(Command): void

<<Abstract>> Command
- receiver: Receiver
+ <<abstract>> execute(): void

Reciever
- player1: boolean
- boatNum: int
- position: ArrayList<Int>
+ placeBoat():void
+ attack(): void

<<abstract>> Listener
+ <<abstract>> takeInData(String strIn): void

Event Publisher
- <<static>> EventPublisher pub
+ <<static>> postEvent(String): void
+ <<static>> subscribe(Listener listener, ArrayList<String> events): void
+ <<static>> unsubscribe(Listener, listener): void

GameRecorder
- filename: String
+ createNewFile(String): void
+ changeFile(String): void
+ update(String): void

Fight
+ execute(): void

Setup
+ execute(): void